mpatrol

Plug Those Memory Leaks

Memory leaks in programs can often cause serious problems ranging from application malfunction to degraded system performance. An open source debugging library, *mpatrol*, helps you handle them effectively.

Detecting memory leaks in C/C++ programs is a challenging task even for experienced programmers. The level of complexity increases when the program to be debugged for memory leaks becomes large or is multi-threaded. Also, memory can present serious problems for programs that run for long periods in the system. Memory leaks can degrade system performance and may lead to system or application crashes. So, software programmers must make sure that their programs do not have memory leaks.

Failing to release dynamically allocated memory will lead to memory leaks. Memory leaks can happen in many forms and may go unnoticed with simple test cases. The following examples show how a program can leak memory.

The program in Figure 1 allocates a total of 1500 bytes of memory in two different calls to *malloc*, but it frees only 1000 bytes, so leaking 500 bytes of memory. The memory leak occurs because the program uses the same pointer variable, *ptr*, to store the address returned by two malloc function calls, and therefore loses the reference to the previously allocated memory.

The program in Figure 3 shows another piece of code that dynamically allocates an array of character pointers to hold five strings, each of the size of 255 bytes. The program erroneously releases only the memory allocated for the array and not the memory allocated for the individual elements. It leaks a total of 1275 (5 × 255) bytes of memory.

Now, let's see how you can detect memory leaks using the free open source memory debugging library, *mpatrol*.

mpatrol is known to work on many platforms, including Linux, Windows, Solaris, AIX and HP-UX. *mpatrol* works by replacing the standard memory allocation functions like *malloc*, *new*, *free*, *delete*, etc, with *mpatrol* equivalents. It can be easily integrated without re-compiling or changing the source code of your program, by dynamically linking the *mpatrol* library during the runtime. It also supports debugging multi-threaded programs.

mpatrol can log all the memory allocations, re-allocations and de-allocations that your program performs into a log file in ASCII text format, which can later be processed by the *mleak* tool, to detect any memory leaks. On systems that support stack trace, *mpatrol* also prints the stack trace along with every log entry. This will greatly help you to easily locate the section of code that leaks memory and find the solution. If performance is critical, then *mpatrol* can be instructed to do tracing instead of logging, which can later be processed by mtrace.

mpatrol can also profile all the memory allocation and de-allocation that your program does. It summarises all the information collected and creates a profiling file that can later be processed with the *mprof* command. This profiling information helps the programmer know the functions that have allocated memory and the details about the unfreed memory allocations that were present at the end of the program execution, some of which could be the